MnistdigitCNN

November 14, 2023

[10]:
```
import tensorflow as tf #import the required libraries
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
```

[3]:
```
(X_train, y_train), (X_test, y_test) = mnist.load_data() #Load the MNIST␣
↪dataset:
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434 [==============================] - 2s 0us/step
```

[4]:
```
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1)#Preprocess the data
X_train = X_train.astype('float32') / 255
X_test = X_test.astype('float32') / 255
```

[5]:
```
y_train = tf.keras.utils.to_categorical(y_train, 10)#Convert the target␣
↪variable to one-hot encoded format:
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

[6]:
```
model = Sequential()#Define the architecture
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

```
Metal device set to: Apple M1

2023-11-14 20:27:01.128053: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:305]
Could not identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel
may not have been built with NUMA support.
2023-11-14 20:27:01.128621: I
tensorflow/core/common_runtime/pluggable_device/pluggable_device_factory.cc:271]
Created TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0
```

```
MB memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id:
<undefined>)
```

[7]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy',␣
↪metrics=['accuracy'])#Compile the model
```

[8]:
```
model.fit(X_train, y_train, epochs=5, batch_size=64, validation_data=(X_test,␣
↪y_test))#Train the model
```

```
Epoch 1/5

2023-11-14 20:27:45.500945: W
tensorflow/core/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU
frequency: 0 Hz
2023-11-14 20:27:45.800232: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

938/938 [==============================] - ETA: 0s - loss: 0.1910 - accuracy:
0.9424

2023-11-14 20:27:59.046753: I
tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:113]
Plugin optimizer for device_type GPU is enabled.

938/938 [==============================] - 15s 13ms/step - loss: 0.1910 -
accuracy: 0.9424 - val_loss: 0.0538 - val_accuracy: 0.9822
Epoch 2/5
938/938 [==============================] - 12s 13ms/step - loss: 0.0534 -
accuracy: 0.9833 - val_loss: 0.0465 - val_accuracy: 0.9869
Epoch 3/5
938/938 [==============================] - 12s 12ms/step - loss: 0.0386 -
accuracy: 0.9879 - val_loss: 0.0302 - val_accuracy: 0.9894
Epoch 4/5
938/938 [==============================] - 12s 12ms/step - loss: 0.0285 -
accuracy: 0.9914 - val_loss: 0.0319 - val_accuracy: 0.9886
Epoch 5/5
938/938 [==============================] - 12s 13ms/step - loss: 0.0227 -
accuracy: 0.9928 - val_loss: 0.0399 - val_accuracy: 0.9880
```

[8]:
```
<keras.callbacks.History at 0x28e058970>
```

[9]:
```
test_loss, test_acc = model.evaluate(X_test, y_test)#Evaluate the model:
print('Test accuracy:', test_acc)
```

```
313/313 [==============================] - 2s 6ms/step - loss: 0.0399 -
accuracy: 0.9880
Test accuracy: 0.9880000352859497
```

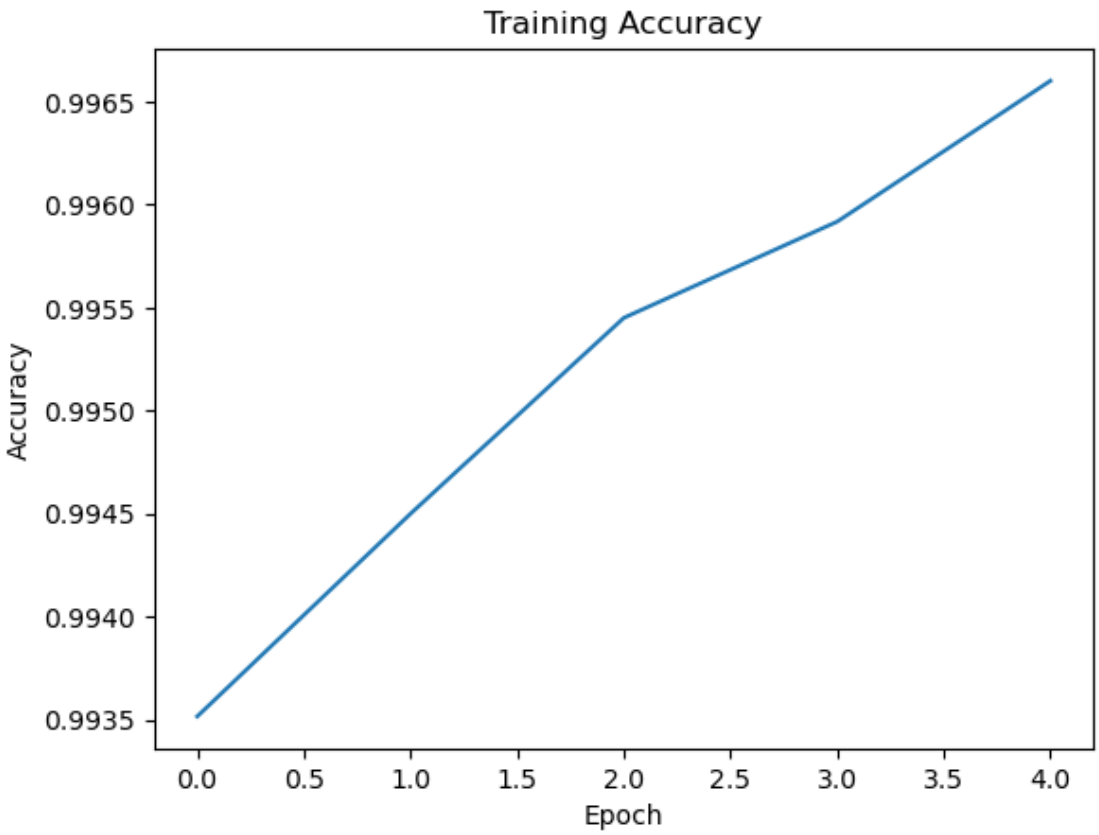
[11]:
```
import matplotlib.pyplot as plt #Import the required library
```

[12]:
```
history = model.fit(X_train, y_train, epochs=5, batch_size=64,␣
↪validation_data=(X_test, y_test))#Train the model while keeping track of the␣
↪training accuracy:
```

```
Epoch 1/5
938/938 [==============================] - 12s 13ms/step - loss: 0.0197 -
accuracy: 0.9935 - val_loss: 0.0271 - val_accuracy: 0.9911
Epoch 2/5
938/938 [==============================] - 12s 13ms/step - loss: 0.0168 -
accuracy: 0.9945 - val_loss: 0.0362 - val_accuracy: 0.9892
Epoch 3/5
938/938 [==============================] - 12s 13ms/step - loss: 0.0138 -
accuracy: 0.9955 - val_loss: 0.0247 - val_accuracy: 0.9927
Epoch 4/5
938/938 [==============================] - 12s 13ms/step - loss: 0.0116 -
accuracy: 0.9959 - val_loss: 0.0327 - val_accuracy: 0.9910
Epoch 5/5
938/938 [==============================] - 12s 13ms/step - loss: 0.0106 -
accuracy: 0.9966 - val_loss: 0.0359 - val_accuracy: 0.9900
```

[13]:
```
train_accuracy = history.history['accuracy']#Access the training accuracy␣
↪values from the history object
```

[14]:
```
plt.plot(train_accuracy)
plt.title('Training Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.show()
```


Training Accuracy
Accuracy
0.9965
0.9960
0.9955
0.9950
0.9945
0.9940
0.9935
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Epoch

[ ]: